

```

// NDDL Grammar (Without semantic checks)

nddl
  :   nddlStatement*
  ;

nddlStatement
  :   typeDefinition
  |   enumDefinition
  |   variableDeclarations
  |   assignment
  |   constraintInstantiation
  |   classDeclaration
  |   allocationStmt
  |   rule
  |   problemStmt
  |   relation
  |   methodInvocation
  |   noopstatement
  ;

enumDefinition
  :   'enum' IDENT enumValues
  ;

enumValues
  :   '{' IDENT (',' IDENT)* '}'
  ;

typeDefinition
  :   'typedef' type baseDomain IDENT ';'
  ;

baseDomain
  :   intervalBaseDomain
  |   enumeratedBaseDomain
  ;

intervalBaseDomain
  :   '[' numericLiteral (',' numericLiteral ']'
  ;

enumeratedBaseDomain
  :   '{' baseDomainValue (',' baseDomainValue)* '}'
  ;

baseDomainValue
  :   literalValue
  |   qualified
  ;

variableDeclarations
  :   ('filter')? type nameWithBaseDomain (',' nameWithBaseDomain)* ';'
  ;

nameWithBaseDomain
  :   (   variable=IDENT ('(' value=initializer ')')?
        |   variable=IDENT '=' value=initializer
        )
  ;

```

```

anyValue
  :   literalValue
  |   baseDomain
  |   qualified
  ;

allocation
  :   'new' constructorInvocation
  ;

constructorInvocation
  :   IDENT variableArgumentList
  ;

qualified
  :   ('this' | IDENT) ('.' IDENT)*
  ;

assignment
  :   qualified ('in' | '=') initializer ';'
  ;

initializer
  :   anyValue
  |   allocation
  ;

classDeclaration
  :   'class' c=IDENT
      (
        (('extends' x=IDENT)? classBlock)
        |
        ';'
      )
  ;

classBlock
  :   '{' classStatement* '}'
  ;

classStatement
  :   variableDeclarations
  |   constructor
  |   predicate
  |   noopstatement
  ;

constructor
  :   IDENT constructorParameterList constructorBlock
  ;

constructorBlock
  :   '{' constructorStatement* '}'
  ;

constructorStatement
  :   assignment
  |   superInvocation
  |   noopstatement
  ;

constructorParameterList
  :   '(' constructorParameters? ')'

```

```

;

constructorParameters
  :      constructorParameter  (',' constructorParameters)?
  ;

constructorParameter
  :      type IDENT
  ;

predicate
  :      'predicate' IDENT predicateBlock
  ;

predicateBlock
  :      '{' predicateStatement* '}'
  ;

// Note: Allocations are not legal here.
predicateStatement
  :      variableDeclarations
  |      constraintInstantiation
  |      assignment
  ;

rule
  :      IDENT '::' IDENT ruleBlock
  ;

ruleBlock
  :      '{' ruleStatement* '}'
  ;

ruleStatement
  :      relation
  |      variableDeclarations
  |      constraintInstantiation
  |      flowControl
  |      noopstatement
  ;

type
  :      'int'
  |      'float'
  |      'bool'
  |      'string'
  |      IDENT
  ;

relation
  :      (token=IDENT | token='this')? temporalRelation predicateArgumentList ';'
  ;

problemStmt
  :      ('rejectable' | 'goal' | 'fact') predicateArgumentList '!!'
  ;

predicateArgumentList
  :      IDENT
  |      '(' predicateArguments? ')'

```

```

;

predicateArguments
:   predicateArgument (',' predicateArgument)*
;

predicateArgument
:   qualified IDENT?
;

constraintInstantiation
:   IDENT variableArgumentList ';'
;

superInvocation
:   'super' variableArgumentList ';'
;

variableArgumentList
:   '(' ^ variableArguments? ')'
;

variableArguments
:   variableArgument ('!' variableArgument)*
;

variableArgument
:   anyValue
;

typeArgumentList
:   '(' typeArguments? ')'
;

typeArguments
:   typeArgument (',' typeArgument)*
;

typeArgument
:   IDENT
;

flowControl
:   'if' guardExpression ruleBlock 'else' ruleBlock)?
|   'foreach' '(' IDENT 'in' qualified ')' ruleBlock
;

guardExpression
:   '(' anyValue (('==' | '!=') anyValue)? ')'
;

allocationStmt
:   allocation ';'
;

temporalRelation
:   'after'
|   'any'
|   'before'
|   'contained_by'
|   'contains'

```

```

| 'contains_end'
| 'contains_start'
| 'ends'
| 'ends_after'
| 'ends_after_start'
| 'ends_before'
| 'ends_during'
| 'equal'
| 'equals'
| 'meets'
| 'met_by'
| 'parallels'
| 'paralleled_by'
| 'starts'
| 'starts_after'
| 'starts_before'
| 'starts_before_end'
| 'starts_during'
;

literalValue
: booleanLiteral
| numericLiteral
| stringLiteral
;

booleanLiteral
: 'true'
| 'false'
;

numericLiteral
: INT
| FLOAT
| ('+')? ('inf' | 'inff')
| '-inf'
| '-inff'
;

stringLiteral
: STRING
;

methodName
: 'specify'
| 'reset'
| 'constrain'
| 'free'
| 'activate'
| 'merge'
| 'reject'
| 'cancel'
| 'close'
;

methodInvocation
: (qualified '.') methodName variableArgumentList ';'
;

tokenNameList
: '(' (^ (tokenNames)? ')' )'!'

```

```

;
tokenNames
:   IDENT (',' IDENT)*
;

noopstatement
:   ';'
;

INCLUDE :   '#include' WS+ file=STRING

IDENT   :   ('a'..'z'|'A'..'Z'|'_'|'$') ('a'..'z'|'A'..'Z'|'_'|'0'..'9'|'$')*
;

STRING  :   '"' (~('\'|'"') | ESCAPE_SEQUENCE)* '"'
;

fragment ESCAPE_SEQUENCE
:   '\\\' ('b'|'t'|'n'|'f'|'r'|'\'|'"'|'\'|'\')
|   UNICODE_ESC
|   OCTAL_ESC
;

fragment OCTAL_ESC
:   '\\\' ('0'..'3') ('0'..'7') ('0'..'7')
|   '\\\' ('0'..'7') ('0'..'7')
|   '\\\' ('0'..'7')
;

fragment UNICODE_ESC
:   '\\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
;

fragment HEX_DIGIT
:   ('0'..'9'|'a'..'f'|'A'..'F')
;

fragment DIGIT
:   ('0'..'9')
;

INT      :   ('+' | '-')? ('0' | '1'..'9' '0'..'9'*) INT_SUFFIX?
;

fragment INT_SUFFIX
:   ('l'|'L')
;

FLOAT    :   ('+' | '-')? ('0'..'9')+ '.' ('0'..'9')* EXPONENT? FLOAT_SUFFIX?
|   '.' ('0'..'9')+ EXPONENT? FLOAT_SUFFIX?
|   ('0'..'9')+ EXPONENT FLOAT_SUFFIX?
|   ('0'..'9')+ FLOAT_SUFFIX
;

fragment EXPONENT
:   ('e'|'E') ('+'|'-')? ('0'..'9')+
;

fragment FLOAT_SUFFIX
:   ('f'|'F'|'d'|'D')

```

```
;  
COMMENT :      '/' ( options {greedy=false;} : . )* '*'/' {$channel=HIDDEN;}  
;  
LINE_COMMENT  
:      '//' ~('\n'|\r)* '\r'? '\n' {$channel=HIDDEN;}  
;  
WS      :      (' |\r'|\t'|\u000C'|\n') {$channel=HIDDEN;}  
;
```